Lexical Encoding and Decoding Framework for Chatbot Design

Author: Elsonlian Neihguk

About the Author

Elsonlian Neihguk is a researcher in computational linguistics and symbolic systems. With a deep interest in formal methods and lightweight AI design, Elsonlian explores rule-based and mathematically grounded frameworks designed for use even in resource-constrained environments.

Introduction

This book introduces a dual-framework approach to handling lexical data for rule-based chatbot systems. It begins with a robust encoder that maps words to structured numerical representations using letter weights and digit decomposition. Building upon this, the decoder module is designed to invert these mappings and recover candidate words from numerical codes, despite potential collisions or ambiguities.

This approach is particularly effective for applications where full language models may be impractical, allowing for symbolic interpretation, matching, and reasoning in chatbots.

Section I: Encoding Framework

(Refer to the earlier provided encoder chapters, covering Definitions through to Theoretical Properties.)

Chapter 1

Decoder Framework

1.1 Inverse Mapping Objective

Given an encoded value $E(w)$, the decoder aims to recover the most probable source word w such that:

$$w = D(E(w)) \quad \text{where } D : \mathbb{N} \to W$$

Since E is not necessarily injective, this inversion may return multiple candidates or require heuristic disambiguation.

1.2 Digit Decomposition Recovery

Given the encoded number E, decompose it as a sum of weighted base-10 digits:

$$E = \sum_{i=1}^{k} r(\sigma_i) \cdot d_i \cdot 10^{p_i}$$

To recover candidate words:

1. Extract digit-position components (d_i, p_i).

2. Use known rank values $r(\sigma)$ to estimate possible letter values σ.

1.3 Rank Decoding Heuristics

Because multiple letters may have similar ranks, define a tolerance window ε and retrieve all letters σ satisfying:

$$|r(\sigma) - \hat{r}_i| \leq \varepsilon \quad \text{where } \hat{r}_i = \frac{E_i}{d_i \cdot 10^{p_i}}$$

This results in a set of possible letters for each digit-position, forming the basis for candidate word generation.

1.4 Candidate Generation and Validation

Construct the cross-product of possible letters for each position:

$$\mathcal{C} = \sigma_1 \times \sigma_2 \times \cdots \times \sigma_k$$

Then filter by the known corpus W:

$$\mathcal{C}' = \{w \in \mathcal{C} \mid w \in W\}$$

If multiple valid words remain, apply contextual ranking or semantic scoring.

1.5 Example Decoding

Suppose:

$$E(\text{HELLO}) = r(H) \cdot 0 + r(E) \cdot 0 + r(L) \cdot 400 + r(L) \cdot 6000 + r(O) \cdot 80000$$

Given known rank assignments (e.g., $r(H) = 10$, etc.), we reconstruct the word "HELLO" via reverse computation.

1.6 Ambiguity and Fallback Strategy

If decoding yields multiple or no candidates:

- Use corpus frequency data to prioritize likely outcomes.
- Apply probabilistic language modeling (if available).
- Default to a fallback such as “Unknown” or request clarification from the user.

1.7 Decoder Complexity

- Digit decomposition: $O(k)$
- Rank estimation: $O(1)$ per digit
- Candidate generation: $O(n^k)$, where n is the average number of candidate letters per slot
- Total decoding: $O(n^k)$ with filtering

Conclusion

The decoder framework allows for reconstruction of original lexical entries from numeric encodings, using reverse-engineered digit decomposition and rank analysis. Though ambiguity is possible due to the non-injective nature of the encoder, heuristic techniques and corpus constraints enable practical and robust decoding—making it highly effective for symbolic chatbot operations.